Software Tools for Networks

John Silberholz

MIT Operations Research Center

January 15, 2015

Real-World Networks in Operations Research

- Energy
  - Power grids
  - Oil and gas pipelines
- Marketing
  - The Internet
  - Social networks
- Public Sector OR
  - Epidemiologic encounter and movement networks
  - Social networks for policing
  - Road networks for disaster response
- Supply chains
- Transportation
  - Road, rail, and **airline networks**
  - Delivery networks

Module Summary

- **Focus: Software tools for network analysis**
- Data Wrangling to Construct Networks in R
- Visualizing Networks
- Network Metrics
  - Computation
  - Integrating into machine learning models
- Network Resilience
- Community Detection

File Locations

Material	Location
Working Directory	`4-graphs`
Flight Data	`On_Time_On_Time_Performance_2014_9.csv`
Airport Data	`../data/airports.csv`
Slides	`Networks.pdf`
Live coding for section X	`code/sectionX.R`
Starter code for section X	`code/exerciseX_start.R`
Complete code for section X	`code/exerciseX_complete.R`

Networks in R

- Our network: September 2014 flight network
  - Vertices: airports
  - Directed edge (a, b): At least one flight from a to b
- Fairly small network
  - 312 nodes
  - 4,020 directed edges representing 469,489 flights
- We will use `igraph` R package
  - Popular general-purpose network package
  - Sparse (edge list) representation
  - Many built in metrics and algorithms (competitive advantage)
  - Mostly implemented in C $\rightarrow$ efficiency
  - `network` and `sna` popular alternatives

Data Wrangling to Construct a Network

- `graph.data.frame(edges, directed, vertices)`
  - `edges`: Edge data frame; first two columns are endpoints and additional columns are metadata
  - `vertices`: (Optional) vertex data frame; first column is name and additional columns are metadata
- Split-apply-combine to compute edges
  - Split flights on start/end airport pair
  - Compute summary information of flights
  - Combine into data frame describing edges
- Split-apply-combine to compute `vertices`
  - Split flights on start airport
  - Compute summary information of flights
  - Combine into data frame describing vertices

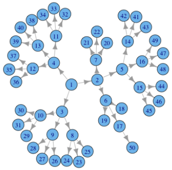
Exercise 1 — Carrier-Specific Flight Networks

- Airline carrier is stored in `Carrier` variable
- Compute `carrier.graphs`, a list with the flight network for each carrier
  - Use `split` to split the data by `Carrier`
  - Use `lapply` with a user-defined function that creates a graph from a data subset
- Starter code in `exercise1_start.R`
- Bonus: Use `vcount` and `graph.density` to find a point-to-point airline and a hub-and-spoke airline
  - Hub-and-spoke have many destinations but few direct flights
  - Point-to-point have fewer destinations and many direct flights

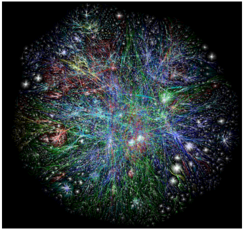
Code	Carrier
AA	American Airlines
AS	Alaska Airlines
B6	JetBlue
DL	Delta
EV	ExpressJet
F9	Frontier Airlines
FL	Airtran
HA	Hawaiian Airlines
MQ	Envoy
OO	SkyWest
UA	United
US	US Airways
VX	Virgin America
WN	Southwest

Visualizing Networks

- Visual representation valuable for understanding networks
- Nodes typically represented by circles
  - Visual properties: size, color, text label
- Edges typically represented by lines
  - Visual properties: width, color, text label, arrowhead
- Algorithmic question: where to plot nodes?
  - **Force-based layout**
  - Spectral layout
  - Arc diagram / hive plot
  - Circular layout
  - ...



Tree graph plotted with `igraph`



Internet graph from `opte.org`

Force-Directed Graph Drawing

- Treat graph as physical system with opposing forces
  - Edges act as springs, pulling vertices together (Hooke's Law)
  - Vertices repel each other, spreading out graph (Coulomb's Law)
- Optimal vertex positioning is nonlinear optimization problem
- Simulated annealing often used to optimize system
- Many similar force-directed layout algorithms in `igraph` (`?igraph::layout`)

Function	**Target Size**
`layout.kamada.kawai`	$\leq$ 100 nodes
`layout.fruchterman.reingold`	100–1000 nodes
`layout.lgl`	$\geq$ 1000 nodes
`layout.drl`	$\geq$ 1000 nodes

Exercise 2 — Manipulating Visual Properties

- Plot the Delta Airlines network (code DL in `carrier.graphs`)
  - Use `layout.lgl` to lay out the graph
  - Node size scales with square root of number of flights from an airport
  - Color Atlanta airport (ATL) red and others black
- Bonus 1: Plot the full network
  - Nodes position from longitude/latitude instead of layout algorithm
  - Adjust `edge.color` to only plot edges with 100 or more flights
  - Mark the top five airports by volume (ATL, ORD, DFW, DEN, LAX) as red and others light gray
- Bonus 2: Replicate Bonus 1, limiting to the continental United States
  - Longitude range $[-130, -60]$
  - Latitude range $[15, 50]$
  - Country "United States"
  - 2:1 width:height ratio for `png` and appropriate `asp` value for `plot`
  - Hint: `?induced.subgraph`

Network Metrics

- Describe structural properties of network
- Vertex metrics
  - `degree(g)`: Number of incident edges
  - `closeness(g)`: Inverse of sum of shortest paths to other nodes
  - `betweenness(g)`: Number of shortest paths containing vertex
  - `page.rank(g)$vector`: Score based on score of linked nodes
  - `transitivity(g, "local")` Probability pair of neighbors connected
- Edge metrics
  - `edge.betweenness(g)`: Number of shortest paths containing edge
  - `degree(g)[get.edges(g, E(g))[,1]]`: Source degree
  - `degree(g)[get.edges(g, E(g))[,2]]`: Sink degree
- Full-network metrics
  - `graph.density(g)`: Proportion of possible edges present
  - `reciprocity(g)`: Proportion of all links that are bidirectional
  - `assortativity.degree(g)`: Correlation of degrees of linked nodes

Exercise 3 — Regression Models over Edges

- Use linear regression to predict the following *edge* outcomes:
  - Average departure delay
  - Average arrival delay
- Use the following edge metrics:
  - Number of flights
  - Edge betweenness
  - Degree of departure and arrival airports
  - PageRank of departure and arrival airports
- Check for multicollinearity between the network metrics

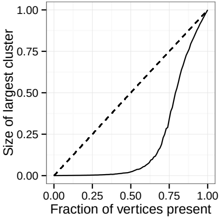
- Bonus: One airport has a relatively low degree (≤ 50) but relatively high betweenness centrality (≥ 5000)
  - Plot degree vs. betweenness to observe this outlier
  - What is the airport?
  - Why does it have this property?
  - Hint: you can access neighbors with `?neighbors`

Network Resilience

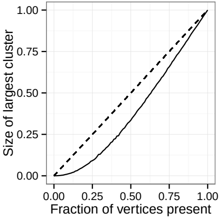
- Study of network properties after removal of vertices or edges
  - *Site percolation*: Remove set of vertices and all connected edges
  - *Bond percolation*: Remove set of edges
- Size of largest component after removing vertices or edges
  - Most commonly studied property
  - Is network performing intended function of connecting vertices?
- Site percolation applications
  - Airport shutdowns due to weather
  - Router failures on the Internet
  - Vaccinating individual within social network
- Bond percolation applications
  - Road failures in disaster response
  - Line failure in telecommunications network or power grid
- Opportunity to compute subgraphs and components in `igraph`

Uniform Random Removal of Vertices

- Retain random proportion Φ of nodes (`induced.subgraph`)
  - Random shutdown of airports
  - Random failure of routers
  - Random vaccination
- Compute size of largest connected component (`cluster`)
- Normalize by vertex count of full graph (`vcount`)
- Simulate multiple random draws, reporting average size (`replicate`)



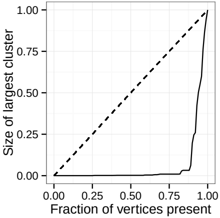
Power grid in western United States



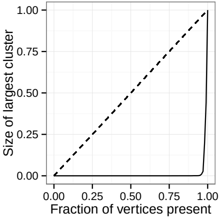
Internet autonomous systems in 2006

Targeted Removal of Vertices

- Retain targeted proportion Φ of nodes with aim to disconnect network
  - Worst-case airport shutdowns
  - Targeted attack on Internet infrastructure
  - Targeted vaccine application
- Computing best fixed-size set NP-hard
- Heuristic approaches used instead
  - Remove nodes with highest degree
  - Remove nodes with highest betweenness
- Compute normalized size of largest component as before



Power grid in western United States



Internet autonomous systems in 2006

Exercise 4 — Bond Percolation

- Perform uniform random bond percolation
  - Randomly retain proportion Φ of *edges* (hint: `?subgraph.edges`)
  - As before, compute normalized size of larges component
  - As before, test for range of Φ values
- Bonus 1: Perform targeted bond percolation
  - Strategy 1: Remove edges with largest minimum degree of endpoints (hint: `?pmin`)
  - Strategy 2: Remove edges with largest edge betweenness
  - Which strategy is most effective?
- Bonus 2: Compare targeted site percolation of Delta (DL) and Southwest (WN) networks

Graph Partitioning and Community Detection

- *Graph Partitioning*: Prespecified structure
  - Input: Number of groups, size of each
  - Output: Graph partition minimizing edge count between groups
  - Example algorithms: Kernighan-Lin, Spectral Partitioning
- *Community Detection*: Find “natural” partitioning
  - No fixed group counts or sizes
  - Multiple definitions of “good partitioning”
  - Many algorithms (`igraph` has nine)
- Market segmentation
  - Groups typically cohesive (many links among members)
  - Groups typically don’t mix (few links between groups)
- Communities easily detached
  - Useful for epidemiologist performing vaccination
  - Area of concern in telecommunication or transportation networks
- Community can be used in prediction algorithms

Modularity Maximization

- Popular community detection objective: modularity
- $Q = \frac{1}{2m}\sum_{i,j}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j)$
  - m: graph edge count
  - A_{ij}: are i and j joined by an edge? (binary)
  - k_i: degree of i
  - $\delta(c_i, c_j)$: are i and j in same partition? (binary)
  - $\frac{k_i k_j}{2m}$: probability i and j would be joined by chance
- Nodes connected at above-chance levels should be in same cluster
- Nodes connected at below-chance levels should be in different clusters
- Optimal number of clusters varies by graph
- Heuristics typically employed to optimize modularity

Exercise 5 — Adding Communities to Prediction Models

- Add departure and arrival community to regression for edge outcomes (Exercise 3)
  - Compute communities with whole graph (not continental U.S.)
  - Model as factor variables (hint: `?as.factor`)
  - Note any changes in adjusted R^2
  - Reminder: `code/exercise3_complete.R`
- Bonus: perform targeted bond percolation using communities
  - Compute indicator for whether each edge bridges communities
  - Order removal priority first by this indicator, then by edge betweenness
  - Compare to targeted strategies from Exercise 4, Bonus 1